PROGRAM CODE

```
#include<stdio.h>
#include<stdlib.h>

#define STATE_UNKNOWN 0
#define STATE_READY 1
#define STATE_RUNNING 2
#define STATE_RETURNED 3

struct entry {
   int AT, BT, CT, TAT, WT, ST, state, rBT;
   char Name[20];
} pChart[10];
int n, readyQue[10], ready_f = 0, ready_r = 0, arrSort[10], tmQntm;
int gEntry[40][2], gTop;

void swap (int* list, int i1, int i2) {
   int temp = list[i1];
   list[i1] = list[i2];
   list[i2] = temp;
}

void enque(int id) {
   if (ready_f-ready_r >= n) { printf("Err: Que overflow\n"); return; }
   pChart[id].state = STATE_READY;
   readyQue[(ready_f++)%n] = id;
}

int nextProcessId() {
   if (ready_r == ready_f) return -1;
   int id = readyQue[ready_r++];
   if (ready_r >= n) { ready_f %= n; ready_r %= n; }
   return id;
}

void printChar(char c, int count) {
   for (int i = 0; i < count; i++) printf("%c", c);
}

int main () {
   printf("Number of Processes >> ");
   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
      printf("Process %d (PID_AT_BT) >> ", i+1);
      scanf("%s %d %d",pChart[i].Name,&pChart[i].AT,&pChart[i].BT);
      arrSort[i] = i;
      pChart[i].state = STATE_UNKNOWN;
      pChart[i].Name[7] = '\0';
   }
   printf("Time Quantum >> ");
```

```
scanf("%d", &tmQntm);

for (int i = 1; i < n; i++)
for (int j = i; j > 0 && pChart[arrSort[j-1]].AT > pChart[arrSort[j]].AT; j--)
   swap(arrSort, j-1, j);

int t_TAT = 0, t_WT = 0, target = 0, cTime = 0, pid = -1;
gTop = 0;
while (target < n || ready_f != ready_r) {
   while (target < n && pChart[arrSort[target]].AT <= cTime) {
      int id = arrSort[target++];
      if (pChart[id].state != STATE_UNKNOWN) continue;
      if (pChart[id].AT > cTime) break;
      pChart[id].rBT = pChart[id].BT;
      pChart[id].state = STATE_READY;
      enque(id);
   }

   if (pid != -1 && pChart[pid].state == STATE_RUNNING) {
      enque(pid);
      pChart[pid].state = STATE_READY;
   }

   pid = nextProcessId();
   struct entry *cp = &pChart[pid];

   if (pid > -1) {
      pChart[pid].state = STATE_RUNNING;
      if (cp->BT == cp->rBT) cp->ST = cTime;
      if (cp->rBT > tmQntm) {
         cp->rBT -= tmQntm;
         cTime += tmQntm;
      } else {
         cTime += cp->rBT;
         cp->rBT = 0;
         cp->CT = cTime;
         cp->TAT = cp->CT - cp->AT;
         cp->WT = cp->TAT - cp->BT;
         t_TAT += cp->TAT;
         t_WT += cp->WT;
         cp->state = STATE_RETURNED;
      }

      gEntry[gTop][0] = pid;
      gEntry[gTop++][1] = cTime;
   } else {
      if (gEntry[gTop-1][0] != -1) gEntry[gTop++][0] = -1;
      cTime++;
   }
}
printf("\n+---------+----+----+----+-----+----+\n");
```

```
    printf("| PROCESS | AT | BT | CT | TAT | WT |\n");
    printf("+---------+----+----+----+-----+----+\n");
    for (int i = 0; i < n; i++) {
        printf("|%9s|%4d|%4d|", pChart[i].Name, pChart[i].AT, pChart[i].BT);
        printf("%4d|%5d|%4d|\n", pChart[i].CT, pChart[i].TAT, pChart[i].WT);
    }
    printf("+---------+----+----+----+-----+----+\n");

    printf("\nAvg TAT = %f\nAvg WT = %f\n", (float)t_TAT/n, (float)t_WT/n);
}
```

OUTPUT